Python for Linux System Adminstration

Vern Ceder
Lead Developer, Zoro Tools, Inc

http://sites.google.com/site/pythonforlinux/

Where we're going...

- Intro
- Basic Python
- Beginning Tricks
- Lunch
- More complex scenarios
- Wrap-up
  - where to go from here
  - resources
  - feedback

How we'll get there...

- audience participation
- a certain amount of my talk
- hands on examples
- (and did I mention audience participation?)

Getting the examples

- Grab a USB drive floating around the room
- Get python_sysadmin_1.tar.gz
- Go ahead and get python_sysadmin_2.tar.gz if you're in the afternoon class
- If you want, get a version of the Python 2.7 docs (or go to http://docs.python.org)
- Unpack and enjoy

Why Python?

- Readable
- Powerful
- Mature
- "Batteries included"
- "low floor, high ceiling"

vs. bash

- bash is a pain when things get complex
- when you start racking your brain, "how can I do this in bash"... it's time for Python

vs. perl

perl has a long history and huge library but:

- reading it makes my head hurt
- perl and objects aren't a smooth fit
- CPAN is a hassle

vs. C/C++

by definition C (and C++) can do anything, but:

- development time is much longer
- required skills are greater
- possibility for nasty bugs is huge

vs. Java

Really? Do I even have to go there?

(but there is Jython)

vs. everybody else...

- Ruby
- Go
- C#
- Lisp/Closure
- Erlang
- Etc, etc, etc...

Python versions

- Python 1.5.2 - still around, but SOOO 90's
- Python 2.4, 2.5 - common, functional but outdated
- Python 2.6, 2.7 - the last of the 2.x series
- Python 3.1 - currently available, libraries being ported
- Python 3.2 – current version of 3.x, the future

On with the show...

Gettting started with Python

You'll need:

- a terminal window and an editor
- your favorite text editor
  - VI (vim)
  - emacs
  - gedit, kate
  - nano
  - IDLE

A first program - motd

```
#!/usr/bin/env python
""" This is a simple motd type program
this is a 'docstring' (which is better than a "comment")
Vern Ceder
09/08/2011
"""
# this is a comment
print "Welcome to python"
```

Pythonic notes - motd01.py

- includes "sh-bang", to make runnable
- `chmod +x motd01.py`
- docstring - comment readable by pydoc
- quotes - triple, double, single
- # for regular comments
- print statement (becomes function in 3.x)

A second version - motd02.py

```
#!/usr/bin/env python
""" This is a motd type program with input
Vern Ceder
09/08/2011
"""
# get user's name
name = raw_input('Enter your name: ')
size = len(name)
print "Welcome to python", name
print "your name is", size, "characters long"
```

Pythonic notes - motd02.py

- `raw_input` to get input (as string) from user (becomes input in Python 3)
- use of variables 'name', 'size'
- `len()` function, for anything with a length
- `print` statement - multiple values, comma

Shell session #1

To run a python shell, just run `python`

- `>>>` and `. . .` prompts
- very handy for experimenting
- accessing docs
- not so good for longer programs

```
>>> 4 + 3
7
>>> a = 4
>>> a
4
>>> a += 4
>>> a
8
```

A bit on Python variables

- ALL variables are references to objects
- variables are created when used
- copying a variable copies the reference
- variables don't have a type, but objects *DO*

A bit on Python variables

```
>>> a = 'one'
>>> b = 2
>>> c = a
>>> a + b ---> type mismatch error
>>> a = 1
>>> a + b
>>> 3
>>> a + c ---> type mismatch error
```

Final motd - import, strings, lists

```
#!/usr/bin/env python
""" This is a motd type program with commandline input
"""
import sys
from datetime import datetime
# get user's name from command line
name = sys.argv[1]
size = len(name)
print "Welcome to python", name
print "The your name is", size, "characters long"
print "The program %s was called with the following %d arguments:" %
(sys.argv[0], len(sys.argv) - 1)
for arg in sys.argv[1:]:
print "
%s" % arg
```

Pythonic notes - motd03.py

- `import, from x import y`
- `sys.argv`
- lists, indexes and slices
- string formatting with %
- `for` loops
  - `break`
  - `continue`
  - `else`
- `while` loops:

```
x = 0
while x < 4:
  print x
  x += 1
```

Shell session #2 - iterables

- range(x) returns a list 0..x-1(not a list in Py3)
- lists vs tuples
- indexes and slices

```
>>> x = [1, 2, 3]
>>> x[1]
2
>>> x[1] = 4
>>> x
[1, 4, 3]
>>> x[1:]
[4, 3]
```

Shell session #2 - iterables, 2

More slicing (works on strings, too!):

```
>>> x[:-1]
[1, 4]
>>> x[1:1] = [5, 6, 7]
>>> x
[1, 5, 6, 7, 4, 3]
>>> x[::2]
[1, 6, 4]
>>> 'abcdef'[0]
'a'
>>> 'abcdef'[-2:]
'ef'
```

Shell session #2 - iterables, 3

range() and tuples

```
>>> range(4)
[0, 1, 2, 3]
>>> x = [1, 2, 3]
>>> y = (1, 2, 3)
>>> y[0]
1
>>> y[0] = 1
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item
assignment
```

Mimicking wc

```
#!/usr/bin/env python
import sys

#data = sys.stdin.read()
data = open(sys.argv[1]).read()
# the above is a shortened form of:
#  infile = open(sys.argv[1])
#  data = infile.read()

chars = len(data)
words = len(data.split())
lines = len(data.split('\n'))

print ("{0}   {1}   {2}".format(lines, words, chars))
#sys.stdout.write(("{0}  {1}  {2}\n".format(lines, words, chars)))
```

Pythonic notes - wc01.py

- opening and reading files
- `split()` on strings
- `format()` (2.6 and up)
- `sys.argv`
- `sys.stdin, sys.stdout, sys.stderr`

Shell session #3 - getting help

dir(), help() and pydoc

```
>>> dir()
>>> dir(sys)
>>> help(sys)
```

```
doc@x60:~$ pydoc sys
doc@x60:~$ pydoc -p 8000
```

(open browser to http://locahost:8000)

Getting more help

- http://docs.python.org
- Downloadable as archive in PDF or HTML (off of USB drive)
- Library documentation
- glossary

Program structure – wc02.py

```
#!/usr/bin/env python
""" wc02.py
uses better structure for scripts
"""
import sys
if __name__ == '__main__':
    data = sys.stdin.read()
    chars = len(data)
    words = len(data.split())
    lines = len(data.split('\n'))
    print ("{0}  {1}  {2}".format(lines, words,chars))
```

Pythonic notes - wc02.py

- `'__main__'` and `__name__`
- re-test with pydoc
- indentation
  - spaces, not tabs
  - 4 spaces

Who's hogging the disk?

```
#!/usr/bin/env python
""" a script to see who's hogging public space """
import sys
def get_totals(rows):
    """ function to find space used by users in a ls -l listing
    rows: ls -l listing as list of lines
    returns: list of user, file space tuples"""
    users = {}
    for row in rows:
        if row.startswith('-'):
            fields = row.split()
            username, filesize = fields[2], fields[4]
            if username not in users:
                users[username] = 0
            users[username] += int(filesize)
    userlist = users.items()
    userlist.sort(cmp=lambda x,y: cmp(x[1], y[1]), reverse=True)
    return userlist
```

Who's hogging the disk? (part 2

```
def compare(x,y):
    return cmp(x[1], y[1])

if __name__ == '__main__':
    file_rows = sys.stdin.readlines()
    user_totals = get_totals(file_rows)
    for user in user_totals:
        print "%-20s %10s" % user
```

Pythonic notes - hogs.py

- dictionaries
  - `keys(), items(), values()` (not lists in 3)
  - `"in"`
- functions
- lambda
- docstrings in functions
- naming conventions
- tuple packing/unpacking
- list sorting (and reversing)
- string formatting - width

Pythonic notes - functions

- form:

```
def func_name(param1, p2=None, [*p3 | **kw]):
    body
    return value (if any)
```

- parameters
  - positional
  - named
  - default values
  - list of remaining positional
  - keyword dict of remaining named
- can return more than one object (tuple packing/unpacking)

function example

```
#!/usr/bin/env python
""" function parameter and return demo """
def foo(a, b, c=None, **kw):
    print "a", a
    print "b", b
    print "c", c
    print "kw", kw
    return a, b
def bar(a, b, *l):
    print "a", a
    print "b", b
    print "l", l
if __name__ == '__main__':
    x, y = foo(1, 2, l=[4,5,6], kw={"key1":7, "key2":8})
    print x,y
    x, y = bar(1, 2, 3, [4,5,6], {"key1":7, "key2":8})
    print x,y
```

Using functions from another file

```
#!/usr/bin/env python
""" use_hogs.py - import a function from another
module"""

from hogs import get_totals

if __name__ == '__main__':
    file_rows = sys.stdin.readlines()
    user_totals = get_totals(file_rows)
    for user in user_totals:
        print "%-20s %10s" % user
```

Yes, we have no switch

It's possible to live without a switch statement:

- `if... elif.... else` works fine
- a dict of functions also works:

Simulating switch

```
#!/usr/bin/env python
""" simulate a switch statement
"""
def one():
    print "You chose number 1"
def two():
    print "You chose number 2"
def three():
    print "You chose number 3"
menu = {'1':one, '2':two, '3': three}

if __name__ == '__main__':
    choice = raw_input("Choose 1, 2, 3 ")
    menu[choice]()
```

Pythonic notes - switch

- everything, even functions, is an object
- objects can be stored in dictionaries and lists
- functions can be called from dictionaries

Calling external commands

```
!/usr/bin/env python
""" execute something outside of Python
"""

import subprocess
retcode = subprocess.call(['ls', '-l', '/home'])
print "returned", retcode
retcode = subprocess.call(['ls', '-l',
'/home/nobody'])
print "returned", retcode
```

Pythonic notes - call

- command and params form list of string elements
- returns 0 on success
- returns process error code on failure, or exception if command not found